C# 프로그래밍

- 6 주차 (2강)

SEJONG UNIVERSITY

- 클래스 간의 형식변환
- 클래스의 읽기 전용 필드
- 분할 클래스
- 확장 메소드
- 구조체

클래스 간의 형식 변환

- 타입을 정의하는 것은 "단위 (unit)"를 사용하는 프로그램에 유용
  - ✓ 예를 들어, 원, 달러, 엔화와 같은 통화 (currency) 단위 사용의 경우
  - ✓ 모든 종류의 통화를 하나의 타입으로 지정하면 금전 계산에 오류 발생의 여지 있음
  - ✓ 아래 코드에서 달러를 엔화에 그대로 대입하면 계산의 오류 발생

```
decimal won = 30000;
decimal dollar = won * 1200;
decimal yen = won * 13;

yen = dollar;   // 실수로 이렇게 대입해도 컴파일 오류를 발생하지 않음
```

클래스 간의 형식 변환

- 원화와 엔화에 대한 클래스 정의
  - ✓ 서로 다른 클래스의 객체를 바로 대입할 수 없음

```
public class Currency
{
    decimal money;
    public decimal Money { get { return money; } }
    public Currency(decimal money)
    {
        this.money = money;
    }
}

public class Won : Currency
{
    public Won(decimal money) : base(money) { }

    public override string ToString()
    {
        return Money + "Won";
    }
}
```

```
public class Yen : Currency
{
    public Yen(decimal money) : base(money) { }
    public override string ToString()
    {
        return Money + "Yen";
    }
}
```

```
static void Main(string[] args)
{
    Won won = new Won(1000);
    Yen yen = new Yen(13);

    won = yen;  // 컴파일 에러
}
```

클래스 간의 형식 변환: implicit 연산자

- 서로 다른 클래스의 객체 사이의 형식 변환 가능
  - ✓ 즉, 두 객체 사이의 대입 연산자 (=) 사용 가능
  - ✓ 암시적, 명시적 형식 변환 모두 가능
  - ✓ 통화 (currency) 예에서 환율을 적용한 계산 가능

```
public class Yen : Currency
{
    public Yen(decimal money) : base(money) { }
    public override string ToString()
    {
        return Money + "Yen";
    }

    static public implicit operator Won(Yen yen)
    {
        return new Won(yen.Money * 13m);
    }
}
```

```
static void Main(string[] args)
{
    Yen yen = new Yen(100);

    Won won1 = yen;      // 암시적(implicit) 형변환
    Won won2 = (Won)yen;// 명시적(explicit) 형변환
    Console.WriteLine(won1 + ", " + won2);
}
```

출력 결과:

```
1300Won, 1300Won
```

클래스 간의 형식 변환: explicit 연산자

- explicit 연산자를 사용해 명시적 형식 변환만 가능하도록 설정 가능

```
public class Dollar : Currency
{
    public Dollar(decimal money): base(money) { }
    public override string ToString()
    {
        return Money + "Dollar";
    }

    static public explicit operator Won(Dollar dollar)
    {
        return new Won(dollar.Money * 1000m);
    }
}
```

```
static void Main(string[] args)
{
    Dollar dollar = new Dollar(1);

    //Won won3 = dollar;  // 암시적 형변환 불가능 (에러발생)
    Won won4 = (Won)dollar; // 명시적 형변환
    Console.WriteLine(won4);// 출력 1000Won
}
```

읽기 전용 필드

- readonly 키워드를 사용하여 정의하는 읽기 전용 필드
  - ✓ 생성자에서 초기화 가능
  - ✓ 초기화 후에는 중간에 값 변경 불가
  - ✓ 생성자 외 다른 메소드에서 값 변경 시 컴파일 에러 발생

```
class Configuration
{
    private readonly int min;
    private readonly int max;

    public Configuration(int v1, int v2)
    {
        min = v1;
        max = v2;
    }

    public void ChangeMax(int newMax)
    {
        // max = newMax;    // 컴파일 에러
    }
}
```

중첩 클래스

- 클래스 안에 클래스 선언
  - ✓ 객체를 생성하고 메소드를 호출하는 방법은 일반 클래스 다르지 않음
  - ✓ 일반 클래스와 차이점은 자신이 소속된 클래스의 멤버에 자유롭게 접근
  - ✓ 자신이 소속된 클래스의 private 멤버에도 접근 가능

```
class OuterClass
{
    private int OuterMember;

    class NestedClass
    {
        public void DoSomething()
        {
            OuterClass outer = new OuterClass();
            outer.OuterMember = 10; // OuterClass 의 private 멤버에 접근 가능
        }
    }
}
```

중첩 클래스 예제코드

```
class Configuration
{
    // List 는 앞으로 배울 새로운 자료구조
    List<ItemValue> listConfig = new List<ItemValue>();

    public void SetConfig(string item, string value)
    {
        ItemValue iv = new ItemValue();
        iv.SetValue(this, item, value);
    }

    public string GetConfig(string item)
    {
        foreach (ItemValue iv in listConfig)
        {
            if (iv.GetItem() == item)
                return iv.GetValue();
        }
        return "";
    }
```

```
    private class ItemValue // 외부에서 접근 불가
    {
        private string item;
        private string value;
        public void SetValue(Configuration config, string
item, string value)
        {
            this.item = item;
            this.value = value;

            bool found = false;
            for (int i=0; i<config.listConfig.Count; i++) {
                if(config.listConfig[i].item == item) {
                    config.listConfig[i] = this;
                    found = true;
                    break;
                }
            }
            if (found == false)
                config.listConfig.Add(this);
        }
        public string GetItem()
        { return item; }
        public string GetValue()
        { return value; }
    }
}
```

중첩 클래스 예제코드

```
static void Main(string[] args)
{
    Configuration Config = new Configuration();
    Config.SetConfig("Version", "V 5.0");
    Config.SetConfig("Size", "655,324 KB");

    Console.WriteLine(Config.GetConfig("Version"));
    Console.WriteLine(Config.GetConfig("Size"));

    Config.SetConfig("Version", "V 5.0.1");
    Console.WriteLine(Config.GetConfig("Version"));
}
```

출력 결과:
V 5.0
655,324 KB
V 5.0.1

분할 클래스

- 여러 번에 나눠서 구현하는 클래스
  - ✓ partial 키워드를 이용
  - ✓ 클래스의 구현이 길어질 경우 여러 파일에 나눠서 구현
  - ✓ 컴파일러는 하나의 클래스로 묶어서 컴파일

```
partial class MyClass
{
    public void Method1() { }
    public void Method2() { }
}

partial class MyClass
{
    public void Method3() { }
    public void Method4() { }
}
```

```
static void Main(string[] args)
{
    MyClass obj = new MyClass();
    obj.Method1();
    obj.Method2();
    obj.Method3();
    obj.Method4();
}
```

확장 메소드

- 기존 클래스의 기능을 확장하는 기법
  - ✓ 예를 들어, string 클래스에 문자열을 뒤집는 기능을 넣을 수 있음
  - ✓ 또한 int 형식에 제곱 연산 기능을 넣을 수도 있음

사용형식

```
namespace 네임스페이스이름
{
    public static class 클래스이름
    {
        public static 반환형식 메소드이름( this 대상형식 식별자, 매개변수목록 )
        {
            //
        }
    }
}
```

```
namespace MyExtension
{
    public static class IntegerExtension
    {
        public static int Power(this int myInt, int exponent)
        {
            int result = myInt;
            for (int i = 1; i < exponent; i++)
                result = result * myInt;
            return result;
        }
    }
}
```

확장 메소드 예제코드

```
using MyExtension;

namespace MyExtension
{
    public static class IntegerExtension
    {
        public static int Square(this int myInt)
        {
            return myInt * myInt;
        }

        public static int Power(this int myInt, int exponent)
        {
            int result = myInt;
            for (int i = 1; i < exponent; i++)
                result = result * myInt;
            return result;
        }
    }
}
```

```
namespace ExtensionMethod
{
    class MainApp
    {
        static void Main(string[] args)
        {
            Console.WriteLine($"3^2 : {3.Square()}");
            Console.WriteLine($"3^4 : {3.Power(4)}");
            Console.WriteLine($"2^10 : {2.Power(10)}");
        }
    }
}
```

출력 결과:

```
3^2 : 9
3^4 : 81
2^10 : 1024
```

구조체

- struct 키워드를 사용하고 클래스와 상당 부분 비슷함
  - ✓ 필드와 메소드를 가짐

특징	클래스	구조체
키워드	class	struct
형식	참조 형식 (힙에 할당)	값 형식 (스택에 할당)
복사	얕은 복사	깊은 복사
인스턴스 생성	new 연산자와 생성자 필요	선언만으로 생성
생성자	매개변수 없는 생성자 선언 가능	매개변수 없는 생성자 선언 불가능
상속	가능	값 형식이므로 상속 불가능

사용형식

```
struct 구조체이름
{
    // 필드
    // 메소드
}
```

```
struct MyStruct
{
    public int MyField1;
    public int MyField2;
    public void MyMethod() { /* ... */ }
}
```

구조체 예제코드

```
struct Point3D
{
    public int X;
    public int Y;
    public int Z;

    public Point3D(int X, int Y, int Z)
    {
        this.X = X;
        this.Y = Y;
        this.Z = Z;
    }

    public override string ToString()
    {
        return string.Format($"{X}, {Y}, {Z}");
    }
}
```

```
static void Main(string[] args)
{
    Point3D p3d1;
    p3d1.X = 10;
    p3d1.Y = 20;
    p3d1.Z = 40;
    Console.WriteLine(p3d1.ToString());

    Point3D p3d2 = new Point3D(100, 200, 300);
    Point3D p3d3 = p3d2;
    p3d3.Z = 400;
    Console.WriteLine(p3d2.ToString());
    Console.WriteLine(p3d3.ToString());
}
```

출력 결과:

```
10, 20, 40
100, 200, 300
100, 200, 400
```

변경 불가능 구조체 선언

- readonly 키워드 사용하여 구조체 선언
  - ✓ 모든 필드와 프로퍼티 값을 수정 할 수 없음
  - ✓ 해당 구조체의 모든 필드가 readonly 로 선언되도록 강제

```
readonly struct ImmutableStruct
{
    public readonly int ImmutableField;
    public ImmutableStruct(int initValue)
    {
        ImmutableField = initValue; // 생성자에서만 초기화 가능
    }
}
```

```
static void Main(string[] args)
{
    ImmutableStruct im = new ImmutableStruct(123);
    im.ImmutableField = 456; // 컴파일 에러
}
```

변경 불가능 구조체 예제 코드

```
readonly struct RGBColor
{
    public readonly byte R;
    public readonly byte G;
    public readonly byte B;

    public RGBColor(byte r, byte g, byte b)
    {
        R = r;
        G = g;
        B = b;
    }
}
```

```
static void Main(string[] args)
{
    RGBColor Red = new RGBColor(255, 0, 0);
    // Red.G = 100; 컴파일 에러
}
```

읽기 전용 메소드

- readonly 키워드 사용하여 메소드 선언
  - ✓ 구조체에서만 선언 가능
  - ✓ 읽기 전용 메소드에서 구조체의 필드를 바꾸려 하면 컴파일 에러 발생

```
struct ACSetting
{
    public double currentInCelsius; // 현재 온도
    public double target; // 희망 온도

    public readonly double GetFahrenheit()
    {
        target = currentInCelsius * 1.8 + 32; // 컴파일 에러
        return target;
    }
}
```

Thank you!